

1. What is Spring Boot and explain its history?

Spring Boot is a project developed on top of Spring Framework. It provides an easier and faster way to setup, configure and run both simple and web-based applications.



In short, Spring Boot is the combination of Spring Framework and Embedded Servers.

It is developed by Pivotal Team. In October 2012, Mike Youngstrom created a feature request in Spring Jira asking for support for containerless web application architectures in Spring Framework. He talked about configuring web container services within a Spring container bootstrapped from the main method! Here is an excerpt from the Jira request.

This request led to the development of the Spring Boot project starting sometime in early 2013. In April 2014, Spring Boot 1.0.0 was released. Current version of Spring Boot is 2.4.3.

2. What are all the advantages of Spring Boot applications?

- ✓ Less configurations.
- ✓ Speed development.
- ✓ XML configuration is not required.
- ✓ Easy to implement the production-ready features like metrics, health checks.
- ✓ Easy integration with any module like JPA.
- ✓ Embedded Servers.
- ✓ YAML Support

3. Explain the disadvantages of Spring Boot?

Spring Boot can use dependencies that are not going to be used in the application. These dependencies increase the size of the application.

4. What is the latest version of Spring Boot and gives its features?

Current version of Spring Boot is 2.4.3

- ✓ Supports Java 8 or above versions
- ✓ Supports Apache Tomcat 8 or above versions
- ✓ Supports Hibernate 5.2

- ✓ JUnit 5's vintage engine is also included by default that supports existing JUnit 4-based test classes. We can also use JUnit 4 and JUnit 5.

5. How Spring Boot will work internally?

Spring Boot annotation internally having the below annotations,

- `@EnableAutoConfiguration` (Load all the dependent beans added by you in build.gradle/pom.xml)
- `@AutoConfiguration` (Initialize all the beans and make those available in IOC container)
- `@ComponentScan` (Responsible for scanning all the classes under base package)

From the run method, the main application context is kicked off which in turn searches for the classes annotated with **@Configuration**, initializes all the declared beans in those configuration classes, and based upon the scope of those beans, stores those beans in JVM, specifically in a space inside JVM which is known as IOC container. After the creation of all the beans, automatically configures the dispatcher servlet and registers the default handler mappings, messageConverters, and all other basic things.

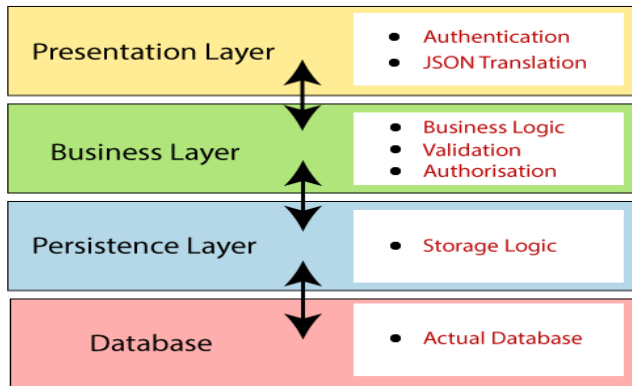
The second class-level annotation is **@EnableAutoConfiguration**. This annotation tells Spring Boot to “guess” how you want to configure spring, based on the jar dependencies that you have added. Since spring-boot-starter-web added Tomcat and Spring MVC, the auto-configuration assumes that you are developing a web application and sets up spring accordingly.

6. Differences b/w Spring, Spring MVC and Spring Boot?

Spring	Spring MVC	Spring Boot
It supports Standalone application	It supports Web application	It support Standalone and web applications
Many configurations required	Many configurations required	Less configurations required
Web.xml not required	Required Web.xml	Web.xml not required
It does not provide support for an in-memory database.	It does not provide support for an in-memory database.	It will provide support for an in-memory database.
It does not provide support for an embedded database.	It does not provide support for an embedded database.	It will provide support for an embedded database.
Developers manually define dependencies for the Spring project in pom.xml.	Developers manually define dependencies for the Spring project in pom.xml.	Spring Boot comes with the concept of starter in pom.xml file that internally takes care of downloading the dependencies JARs based on Spring Boot Requirement.

7. Explain Spring Boot Architecture?

We have four layers in Spring Boot,



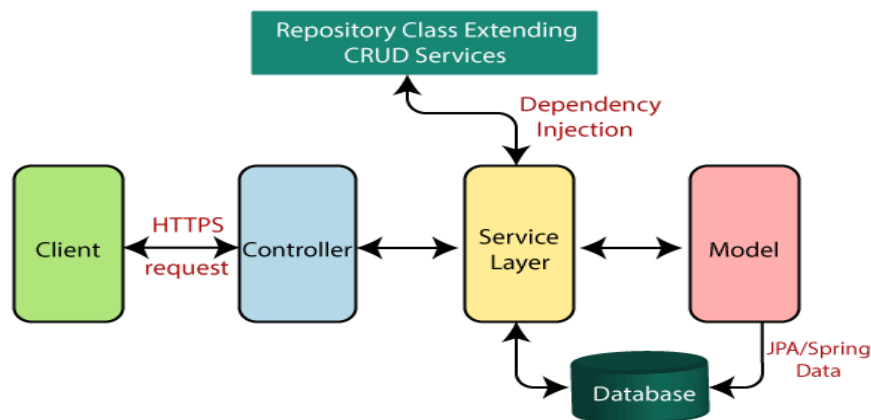
Presentation Layer: The presentation layer handles the HTTP requests, translates the JSON parameter to object, and authenticates the request and transfer it to the business layer. In short, it consists of **views** i.e., frontend part.

Business Layer: The business layer handles all the **business logic**. It consists of service classes and uses services provided by data access layers. It also performs **authorization** and **validation**.

Persistence Layer: The persistence layer contains all the **storage logic** and translates business objects from and to database rows.

Database Layer: In the database layer, **CRUD** (create, retrieve, update, delete) operations are performed.

Spring Boot architecture flow,



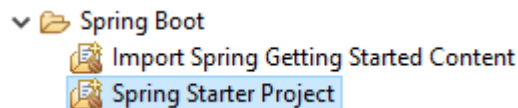
Spring Boot uses all the modules of Spring-like Spring MVC, Spring Data, etc. The architecture of Spring Boot is the same as the architecture of Spring MVC, except one thing: there is no need for DAO and DAOImpl classes in Spring Boot.

- ✓ The client makes the HTTP requests (PUT or GET).
- ✓ The request goes to the controller, and the controller maps that request and handles it. After that, it calls the service logic if required.

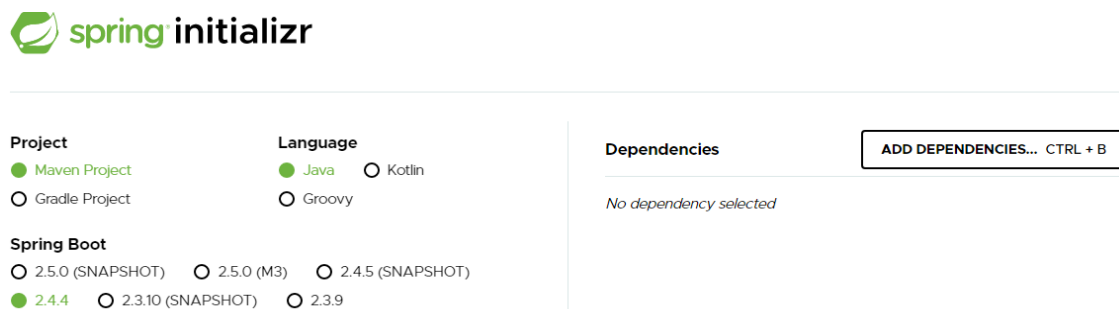
- ✓ In the service layer, all the business logic performs. It performs the logic on the data that is mapped to JPA with model classes.
- ✓ A JSP page is returned to the user if no error occurred.

8. Explain about different ways to create the Spring Boot Project?

STS: Project -> Spring Boot -> Spring Starter Project



Spring initializer: <https://start.spring.io>



Command Line Interface (CLI)

- ✓ Download the CLI zip
- ✓ Setup the extracted CLI folder path in windows environment variables path.
- ✓ Run this command: spring run

9. What is meant by Spring Boot Dependency Management and its advantages?

Spring Boot manages dependencies and configuration automatically. Each release of Spring Boot provides a list of dependencies that it supports. The list of dependencies is available as a part of the Bills of Materials (spring-boot-dependencies) that can be used with Maven. So, we need not to specify the version of the dependencies in our configuration. Spring Boot manages itself. Spring Boot upgrades all dependencies automatically in a consistent way when we update the Spring Boot version.

Advantages of Dependency Management:

- ✓ It provides the centralization of dependency information by specifying the Spring Boot version in one place. It helps when we switch from one version to another.

- ✓ It avoids mismatch of different versions of Spring Boot libraries.
- ✓ We only need to write a library name with specifying the version. It is helpful in multi-module projects.

10. Explain about Spring Initializer?

Spring Initializer is a web-based tool provided by the Pivotal Web Service. With the help of Spring Initializer, we can easily generate the structure of the Spring Boot Project.

It also provides various options for the project that are expressed in a metadata model. The metadata model allows us to configure the list of dependencies supported by JVM and platform versions, etc. It serves its metadata in a well-known that provides necessary assistance to third-party clients.

11. Explain about Spring Boot Starter?

Spring Boot provides a number of starters that allow us to add jars in the classpath. Spring Boot built-in starters make development easier and rapid. Spring Boot Starters are the dependency descriptors.

In the Spring Boot Framework, all the starters follow a similar naming pattern: `spring-boot-starter-*`, where `*` denotes a particular type of application. For example, if we want to use spring and JPA for database access, we need to include the `spring-boot-starter-data-jpa` dependency in our `pom.xml` file of the project.

12. List out few Spring Boot Starter?

`spring-boot-starter-web`
`spring-boot-starter-security`
`spring-boot-starter-mail`
`spring-boot-starter-jpa`
`spring-boot-starter-actuator`
`spring-boot-starter-tomcat`

13. How to add third party starters in project?

We can also include third party starters in our project. But we do not use `spring-boot-starter` for including third party dependency. The `spring-boot-starter` is reserved for official Spring Boot artifacts. The third-party starter starts with the name of the project. For example, the third-party project name is `abc`, then the dependency name will be `abc-spring-boot-starter`.

14. Explain about Spring Boot Starter Parent?

The spring-boot-starter-parent is a project starter. It provides default configurations for our applications. It is used internally by all dependencies. All Spring Boot projects use spring-boot-starter-parent as a parent in pom.xml file.

The spring-boot-starter-parent inherits dependency management from spring-boot-dependencies. We only need to specify the Spring Boot version number.

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.4.4</version>
  <relativePath /> <!-- lookup parent from repository -->
</parent>
```

15. Can we run the spring boot starter without parent?

Yes we can. In some cases, we need not to inherit spring-boot-starter-parent in the pom.xml file. To handle such use cases, Spring Boot provides the flexibility to still use the dependency management without inheriting the spring-boot-starter-parent. We need to specify the version.

```
<!-- Import dependency management from Spring Boot -->
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-dependencies</artifactId>
<version>2.1.1.RELEASE</version>
```

16. Explain about Spring Boot Starter Web?

There are two important features of spring-boot-starter-web:

- ✓ It is compatible for web development
- ✓ Auto configuration

Spring web uses Spring MVC, REST and Tomcat as a default embedded server. The single spring-boot-starter-web dependency transitively pulls in all dependencies related to web development. It also reduces the build dependency count.

17. Explain about Spring Boot Starter Actuator?

Spring Boot Actuator is a sub-project of the Spring Boot Framework. It includes a number of additional features that help us to monitor and manage the Spring Boot application. If we want to get production-ready features in an application, we should use the Spring Boot actuator.

This starter will provide the following features:

- ✓ Health Check
- ✓ Metrics
- ✓ Audit

18. What are all the embedded servers supported by Spring Boot?

Each Spring Boot application includes an embedded server. Embedded server is embedded as a part of deployable application. The advantage of embedded server is, we do not require pre-installed server in the environment. With Spring Boot, default embedded server is **Tomcat**. Spring Boot also supports another two embedded servers:

Jetty Server

Undertow Server

19. How to use the different embedded servers other than tomcat?

With Spring Boot, default embedded server is **Tomcat**. If we want to replace with Jetty/Undertow we just need to place the corresponding dependency pom.xml/build.gradle.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-undertow</artifactId>
</dependency>
```

20. How to update the server port for embedded server?

We can update it on properties/yml file by adding below property,

```
server.port=8085
```

21. In Spring Boot how to use the external servers instead of embedded?

- ✓ In pom.xml, add required server dependency and packaging to war

```
<!-- to export as WAR -->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-tomcat</artifactId>
  <scope>provided</scope>
</dependency>

<!-- packaging to WAR -->
<packaging>war</packaging>
```

- ✓ If you are using the tomcat as external server some of your spring boot libraries contain it by default, exclude it.

```

<dependency>
<groupid>org.springframework.boot</groupid>
<artifactid>spring-boot-starter-web</artifactid>
    <exclusions>
        <exclusion>
            <groupid>org.springframework.boot</groupid>
            <artifactid>spring-boot-starter-tomcat</artifactid>
        </exclusion>
    </exclusions>
</dependency>

```

- ✓ Extend your main class with `SpringBootServletInitializer` and override its `configure` method. It will read your configuration file from an external source where it operation can handle it easily.

```

@SpringBootApplication
public class SpringBootApp extends SpringBootServletInitializer {

    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {
        return application.sources(SpringBootApp.class);
    }

    public static void main(String[] args) {
        SpringApplication.run(SpringBootApp.class, args);
    }
}

```

- ✓ Generate WAR and deploy into the external server

22. Explain endpoints available in Actuator?

End point	Description
info	It is used to display arbitrary application info.
env	Display the current working environment
health	Provides the application status like up or down
beans	It is used to display a complete list of all the Spring beans in your application.
dump	It is used to perform a thread dump
metrics	It is used to show metrics information for the current application.
mappings	It is used to display a collated list of all <code>@RequestMapping</code> paths.
trace	It is used to display trace information.
loggers	It is used to show and modify the configuration of loggers in the application.

23. How to enable all endpoints of Actuator?

We can enable all actuator endpoints by adding below property in application.properties/application.yml

```
management.endpoints.web.exposure.include=*
```

24. How to enable particular endpoint of actuator?

```
management.endpoints.web.exposure.include=health,info,beans,env
```

25. How to secure the actuator endpoints?

Spring Boot enables security for all actuator endpoints. It uses **form-based** authentication that provides **user Id** as the user and a randomly generated **password**. We can also access actuator-restricted endpoints by customizing basicauth security to the endpoints. We need to override this configuration by **management.security.roles** property. For example:

```
management.security.enabled=true
management.security.roles=ADMIN
security.basic.enabled=true
security.user.name=admin
security.user.password=admin
```

26. How add the custom endpoints with actuator?

Spring Boot 2 provides an easy way to create custom endpoints. Spring Boot 2.x introduced **@Endpoint** annotation. Spring Boot automatically expose endpoints with **@Endpoint**, **@WebEndpoint**.

Spring Boot 2.x Actuator support CRUD model, it supports read, writes and delete operation with the endpoints. The **@Endpoint** annotation can be used in combination with **@ReadOperation**, **@WriteOperation** and **@DeleteOperation** to develop endpoints.

```
@Component
@Endpoint(id="custom-health")
public class CustomHealthEndPoint {

    @ReadOperation
    public CustomHealth health() {
        Map<String, Object> details = new LinkedHashMap<>();
        details.put("CustomHealthStatus", "Everything looks good");
        CustomHealth health = new CustomHealth();
        health.setHealthDetails(details);
        return health;
    }
}
```

To access our custom endpoint, use <http://host:port/actuator/custom-health> to check the output.

```
{  
  "CustomHealthStatus": "Everything looks good"  
}
```

27. How to enable or disable the actuator endpoints?

You can enable or disable an actuator endpoint by setting the property `management.endpoint.<id>.enabled` to true or false (where id is the identifier for the endpoint).

```
endpoints.autoconfig.enabled=false  
endpoints.beans.enabled=false  
endpoints.configprops.enabled=false  
endpoints.dump.enabled=false  
endpoints.env.enabled=false  
endpoints.health.enabled=true  
endpoints.info.enabled=true  
endpoints.metrics.enabled=false  
endpoints.mappings.enabled=false  
endpoints.shutdown.enabled=false  
endpoints.trace.enabled=false
```

28. What is the use of shutdown in actuator?

Spring Boot Actuator comes with many production-ready features which include /shutdown endpoint. By default, all /shutdown endpoint is not enabled in the Actuator.

For enabling we need to add below in properties file,

```
management.endpoint.shutdown.enabled=true
```

To test the shutdown endpoint we need to hit the below url it is an post url,

<http://localhost:8080/actuator/shutdown>

It will gracefully shutdown the Spring Bot application.

29. How to disable the Auto Configuration classes?

We can also disable the specific auto-configuration classes, if we do not want to be applied. We use the exclude attribute of the annotation `@EnableAutoConfiguration` to disable the auto-configuration classes

```
@Configuration(proxyBeanMethods = false)
@EnableAutoConfiguration(exclude={DataSourceAutoConfiguration.class})
public class MyConfiguration
{
}
```

30. How to enable the H2 database support in Spring Boot?

We need to add the below property in application.properties file,

```
application.properties

# Enabling H2 Console
spring.h2.console.enabled=true
```

31. What is the use of Spring Boot Thymeleaf?

Thymeleaf is a Java-based library used to create a web application. It provides a good support for serving a XHTML/HTML5 in web applications.

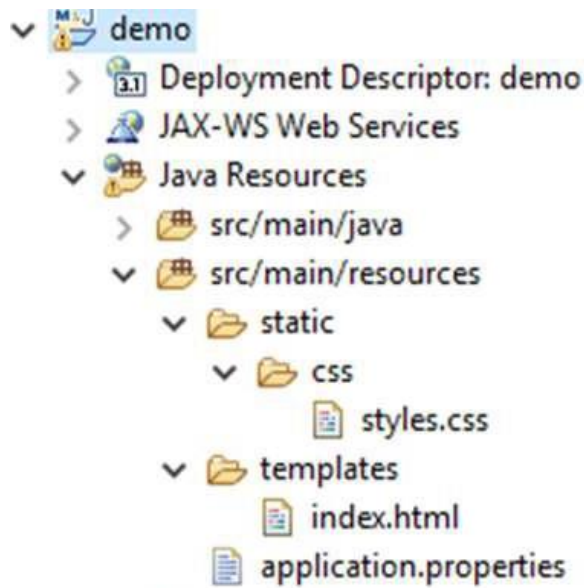
Thymeleaf Templates

Thymeleaf converts your files into well-formed XML files. It contains 6 types of templates as given below –

- XML
- Valid XML
- XHTML
- Valid XHTML
- HTML5
- Legacy HTML5

All templates, except Legacy HTML5, are referring to well-formed valid XML files. Legacy HTML5 allows us to render the HTML5 tags in web page including not closed tags.

We need to add the HTML and CSS is below path of the project to identify them by Thymeleaf,



32. How to implement the Global Exception handling in Spring Boot?

We need to create a controller add the class level annotation **@ControllerAdvice** and method level annotation **@ExceptionHandler**.

@ControllerAdvice: is an annotation, to handle the exceptions globally.

@ExceptionHandler: is an annotation used to handle the specific exceptions and sending the custom responses to the client.

First we need to create the custom exception,

```
public class RecordNotFoundException extends RuntimeException
{
    public RecordNotFoundException(String exception) {
        super(exception);
    }
}
```

Create a bean that will having fields with user understandable details

```
@Data
@NoArgsConstructor
@AllArgsConstructor
public class ExceptionDetails {
    private String message;
    private LocalDateTime localDateTime;
    private String details;
}
```

Then create the Controller to handle the known and unknown exceptions,

```

@ControllerAdvice
public class RegistrationExceptionHandler extends ResponseEntityExceptionHandler {

    @ExceptionHandler(RegistrationException.class)
    public ResponseEntity<Object> handleResourceNotFoundException(RegistrationException foundException,
        WebRequest webRequest) {
        ExceptionDetails exceptionDetails = new ExceptionDetails();
        exceptionDetails.setLocalDateTime(LocalDateTime.now());
        exceptionDetails.setMessage(foundException.getMessage());
        exceptionDetails.setDetails(webRequest.getDescription(false));
        return new ResponseEntity<>(exceptionDetails, HttpStatus.NOT_FOUND);
    }

    @ExceptionHandler(Exception.class)
    public ResponseEntity<Object> handleGlobalException(Exception exception, WebRequest webRequest) {
        ExceptionDetails exceptionDetails = new ExceptionDetails();
        exceptionDetails.setLocalDateTime(LocalDateTime.now());
        exceptionDetails.setMessage(exception.getMessage());
        exceptionDetails.setDetails(webRequest.getDescription(false));
        return new ResponseEntity<>(exceptionDetails, HttpStatus.INTERNAL_SERVER_ERROR);
    }
}

```

In the above example Controller class extending `ResponseEntityExceptionHandler`.

`ResponseEntityExceptionHandler` is a convenient base class for to provide centralized exception handling across all `@RequestMapping` methods through `@ExceptionHandler` methods. `@ControllerAdvice` is more for enabling auto-scanning and configuration at application startup.

33. How to resolve the cross origin issues in Spring Boot?

For example, your web application is running on 8080 port and by using JavaScript you are trying to consuming Restful web services from 9090 port. Under such situations, you will face the Cross-Origin Resource Sharing security issue on your web browsers.

We need to set the origins for RESTful web service by using `@CrossOrigin` annotation for the controller method. This `@CrossOrigin` annotation supports specific REST API, and not for the entire application.

```

@RequestMapping(value = "/products")
@CrossOrigin(origins = "http://localhost:8080")

public ResponseEntity<Object> getProduct() {
    return null;
}

```

Global Cross origin issue solving,

To enable CORS for the whole application, use `WebMvcConfigurer` to add `CorsRegistry`.

```

@Configuration
@EnableWebMvc
public class CorsConfiguration implements WebMvcConfigurer
{
    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**")
            .allowedMethods("GET", "POST");
    }
}

```

34. How to integrate the JPA with Spring Boot?

We need to add the below dependencies in pom.xml/build.gradle,

- ✓ spring-boot-starter-jpa
- ✓ mysql-connector-java (For mysql database)

Add the below properties in application.properties/application.yml file,

```

spring:
  application:
    name: student-teacher-registration-service
  datasource:
    url: jdbc:mysql://localhost:3306/grade_management_system
    username: root
    password: root

  jpa:
    hibernate:
      ddl-auto: update
    properties:
      hibernate:
        dialect: org.hibernate.dialect.MySQL5Dialect
    show-sql: true

```

Create a Repository interface that extends JpaRepository,

```

public interface RegistrationRepository extends JpaRepository<RegistrationEntity, Long> {

```

Autowire the above repository in Controller/Service and use the existing methods of it.

35. How to read the properties from application.properties/application.yml file in Spring Boot?

Added the below property in application.properties file,

```

jwt.secretKey = dacebf30-1467-45d9-8260-5df190407f5f

```

Accessing the above property,

```

@Value("${jwt.secretKey}")
private String secret;

```

36. Difference between application.properties and application.yml?

YML	Properties
Hierarchical Structure	Non-Hierarchical Structure
Spring Framework doesn't support @PropertySources with .yml files	Spring supports @PropertySources with .properties file
If you are using spring profiles, you can have multiple profiles in one single .yml file	Each profile need one separate .properties file
While retrieving the values from .yml file we get the value as whatever the respective type (int, string etc.) is in the configuration	While in case of the .properties files we get strings regardless of what the actual value type is in the configuration
Its usage is quite prevalent in many languages like Python, Ruby, and Java	It is primarily used in java
Supports key/val, basically map, List and scalar types (int, string etc.)	Supports key/val, but doesn't support values beyond the string

37. What is DevTools in Spring Boot?

DevTools stands for Developer Tool. The aim of the module is to try and improve the development time while working with the Spring Boot application. Spring Boot DevTools pick up the changes and restart the application.

38. How do you Add, Filter to an application?

A filter is an object used to intercept the HTTP requests and responses of your application. By using filter, we can perform two operations at two instances –

- ✓ Before sending the request to the controller
- ✓ Before sending a response to the client.

```
@Component
public class FirstFilter implements Filter{

    //this method will be called by container when we send any request
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain){

        System.out.println("doFilter() method is invoked");
        HttpServletRequest httpRequest = (HttpServletRequest)request;
        HttpServletResponse httpResponse = (HttpServletResponse)response;
        System.out.println("Context path is " + httpRequest.getContextPath());
        chain.doFilter(httpRequest, httpResponse);
        System.out.println("doFilter() method is ended");

    }

    // this method will be called by container while deployment
    public void init(FilterConfig config) throws ServletException {

        System.out.println("init() method has been get invoked");
        System.out.println("Filter name is "+config.getFilterName());
        System.out.println("ServletContext name is"+config.getServletContext());
        System.out.println("init() method is ended");

    }

    public void destroy() {
        //do some stuff like clearing the resources
    }
}
```

39. What is the use of FilterRegistrationBean?

FilterRegistrationBean registers a Filter as spring bean and it provides methods to add URL mappings. When we want a filter to only apply to certain URL patterns then will use FilterRegistrationBean. We can set the filter order also.

```
@Configuration
public class WebConfig {
    //Register ABCFilter
    @Bean
    public FilterRegistrationBean<ABCFilter> abcFilter() {
        FilterRegistrationBean<ABCFilter> filterRegBean = new FilterRegistrationBean<>();
        filterRegBean.setFilter(new ABCFilter());
        filterRegBean.addUrlPatterns("/app/*");
        filterRegBean.setOrder(Ordered.LOWEST_PRECEDENCE -1);
        return filterRegBean;
    }
}
```

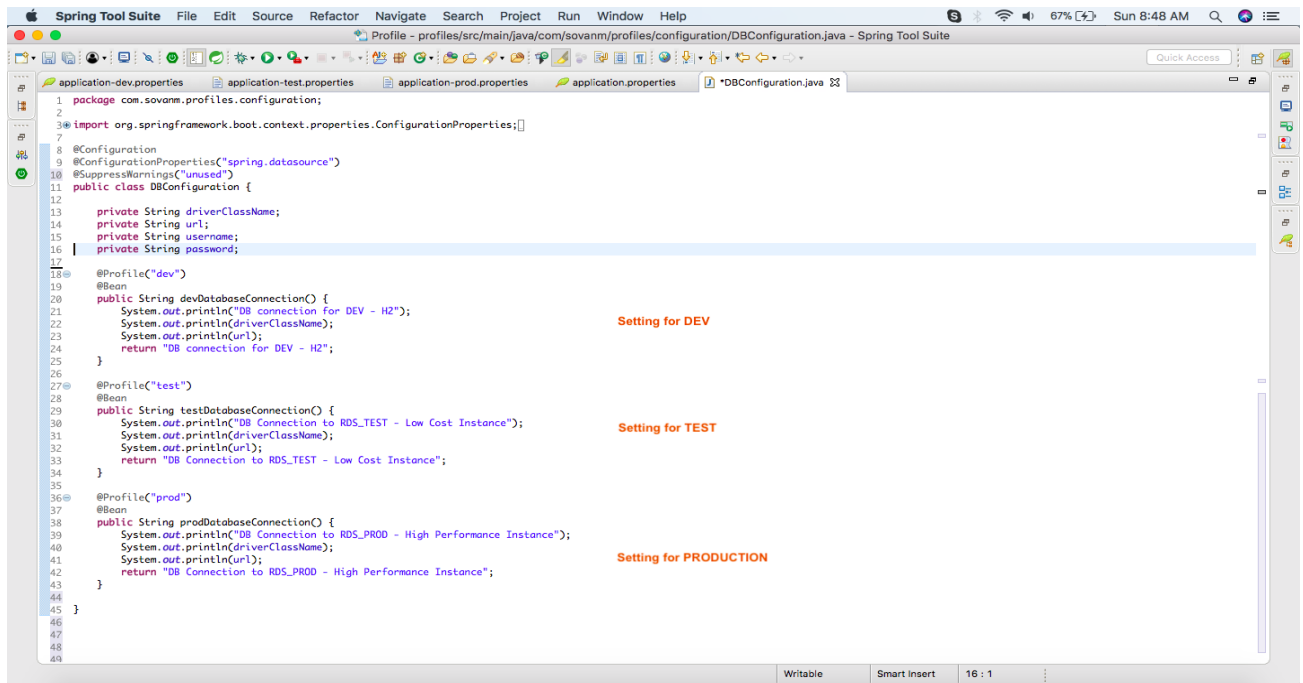
40. What is the use of profiles in Spring Boot?

Profiles are a core feature of the framework — allowing us to map our beans to different profiles — for example, dev, test, and prod.

Each environment requires a setting that is specific to them. For example, in DEV, we do not need to constantly check database consistency. Whereas in TEST and STAGE, we need to. These environments host specific configurations called Profiles.

We will use the application.properties to use the key below:

```
spring.profiles.active=dev
```

We have used the `@Profile("Dev")` to let the system know that this is the BEAN that should be picked up when we set the application profile to DEV. The other two beans will not be created at all.

41. Explain how to register a custom auto-configuration?

In order to register an auto-configuration class, you have to mention the fully-qualified name under the `@EnableAutoConfiguration` key META-INF/spring.factories file. Also, if we build with maven, then this file should be placed in the resources/META-INF directory.

42. How to Deploy Spring Boot Web Applications as Jar and War Files?

We need to add spring-boot-maven-plugin, to package a web application as an executable JAR. To include this plugin, just add a plugin element to pom.xml:

```

<plugin>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-maven-plugin</artifactId>
</plugin>

```

With this plugin in place, we'll get a fat JAR after executing the package phase. This JAR contains all the necessary dependencies, including an embedded server. Thus, we no longer need to worry about configuring an external server.

We can then run the application just like we would an ordinary executable JAR.

Notice that the packaging element in the pom.xml file must be set to jar to build a JAR file:

```
<packaging>jar</packaging>
```

If we don't include this element, it also defaults to jar.

In case we want to build a WAR file, change the packaging element to war:

```
<packaging>war</packaging>
```

And leave the container dependency off the packaged file:




```
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-tomcat</artifactId>  
  <scope>provided</scope>  
</dependency>
```

After executing the Maven package phase, we'll have a deployable WAR file.

43. What is the use of @PropertySource?

Spring @PropertySource annotation is used to provide properties file to Spring Environment. This annotation is used with @Configuration classes.

Spring PropertySource annotation is repeatable, means you can have multiple PropertySource on a Configuration class. This feature is available if you are using Java 8 or higher version.

▼  src/main/resources
  db.properties
  root.properties

```
@Configuration
@PropertySource("classpath:db.properties")
@PropertySource("classpath:root.properties")
public class DBConfiguration {

    @Autowired
    Environment env;

    @Bean
    public DBConnection getDBConnection() {
        System.out.println("Getting DBConnection Bean for
App: "+env.getProperty("APP_NAME"));
        DBConnection dbConnection = new
DBConnection(env.getProperty("DB_DRIVER_CLASS"),
env.getProperty("DB_URL"), env.getProperty("DB_USERNAME"),
env.getProperty("DB_PASSWORD").toCharArray());
        return dbConnection;
    }
}
```